Network Working Group
Request for Comments 746
NIC 43976

Richard Stallman
MIT-AI
17 March 1978

The SUPDUP Graphics Extension

... extends SUPDUP to permit the display of drawings on the screen of the terminal, as well as text. We refer constantly to the documentation of the SUPDUP protocol, described by Crispin in RFC 734 "SUPDUP Protocol".

Since this extension has never been implemented, it presumably has some problems. It is being published to ask for suggestions, and to encourage someone to try to bring it up.

The major accomplishments are these:

* It is easy to do simple things.

* Any program on the server host can at any time begin outputting pictures. No special preparations are needed.

* No additional network connections are needed. Graphics commands go through the normal text output connection.

* It has nothing really to do with the network. It is suitable for use with locally connected intelligent display terminals in a terminal-independent manner, by programs which need not know whether they are being used locally or remotely. It can be used as the universal means of expression of graphics output, for whatever destination. Programs can be written to use it for non-network terminals, with little loss of convenience, and automatically be usable over the ARPA network.

* Loss of output (due, perhaps, to a "silence" command typed by the user) does not leave the user host confused.

* The terminal does not need to be able to remember the internal "semantic" structure of the picture being displayed, but just the lines and points, or even just bits in a bit matrix.

* The server host need not be able to invoke arbitrary terminal-dependent software to convert a standard language into one that a terminal can use. Instead, a standard language is defined which all programmable terminals can interpret easily. Major differences between terminals are catered to by conventions for including enough redundant information in the output stream that all types of terminals will have the necessary information available when it is needed, even if they

are not able to remember it in usable form from one command to another.

Those interested in network graphics should read about the Multics Graphics System, whose fundamental purpose is the same, but whose particular assumptions are very different (although it did inspire a few of the features of this proposal).

SUPDUP Initial Negotiation:

One new optional variable, the SMARTS variable, is defined. It should follow the other variables sent by the SUPDUP user process to the SUPDUP server process. Bits and fields in the left half-word of this variable are given names starting with "%TQ". Bits and fields in the right half are given names starting with "%TR". Not all of the SMARTS variable has to do with the graphics protocol, but most of it does. The %TQGRF bit should be 1 if the terminal supports graphics output at all.

Invoking the Graphics Protocol:

Graphics mode is entered by a %TDGRF (octal 231) code in the output stream. Following characters in the range 0 - 177 are interpreted according to the graphics protocol. Any character 200 or larger (a %TD code) leaves graphics mode, and then has its normal interpretation. Thus, if the server forgets that the terminal in graphics mode, the terminal will not long remain confused.

Once in graphics mode, the output stream should contain a sequence of graphics protocol commands, each followed by its arguments. A zero as a command is a no-op. To leave graphics mode deliberately, it is best to use a %TDNOP.

Co-ordinates:

Graphics mode uses a cursor position which is remembered from one graphics command to the next while in graphics mode. The graphics mode cursor is not the same one used by normal type-out: Graphics protocol commands have no effect on the normal type-out cursor, and normal type-out has no effect on the graphics mode cursor. In addition, the graphics cursor's position is measured in dots rather than in characters. The relationship between the two units (dots, and characters) is recorded by the %TQHGT and %TQWID fields of the SMARTS variable of the terminal, which contain the height and width in dots of the box occupied by a character. The size of the screen in either dimension is assumed to be the length of a character box times the number of characters in that direction on the screen. If the screen is actually bigger than that, the excess is may or may not be part of the visible area; the program will not know that it exists, in any case.

Each co-ordinate of the cursor position is a 14-bit signed number, where zero is at the center of the screen (if the screen dimension is an even number of dots, then the visible negative points extend one unit farther that the positive ones, in proper two's complement fashion). Excessively large values of the co-ordinates will be off the screen, but are still meaningful.

An alternate mode is defined, which some terminals may support, in which virtual co-ordinates are used. The specified co-ordinates are still 14-bit signed numbers, but instead of being in units of physical dots on the terminal, it is assumed that +4000 octal is the top of the screen or the right edge, while -4000 octal is the bottom of the screen or the left edge. The terminal is responsible for scaling these virtual co-ordinates into units of screen dots. Not all terminals need have this capability; the %TQVIR bit in the SMARTS variable indicates that it exists. To use virtual co-ordinates, the server should send a %GOVIR; to use physical co-ordinates again, it should send a %GOPHY. These should be repeated at intervals, such as when graphics mode is entered, even though the terminal must attempt to remember the state of the switch anyway. This repetition is so that a loss of some output will not cause unbounded confusion.

The virtual co-ordinates are based on a square. If the visible area on the terminal is not a square, then the standard virtual range should correspond to a square around the center of the screen, and the rest of the visible area should correspond to virtual co-ordinates just beyond the normally visible range.

Graphics protocol commands take two types of cursor position arguments, absolute ones and relative ones. Commands that take address arguments generally have two forms, one for each type of

address. A relative address consists of two offsets, delta-X and delta-Y, from the old cursor position. Each offset is a 7-bit two's complement number occupying one character. An absolute address consists of two co-ordinates, each 14 bits long, occupying two characters, each of which conveys 7 bits. The X co-ordinate or offset precedes the Y. Both types of address set the running cursor position which will be used by the next address, if it is relative. It is perfectly legitimate for parts of objects to go off the screen. What happens to them is not terribly important, as long as it is not disastrous, does not interfere with the reckoning of the cursor position, and does not cause later objects, drawn after the cursor moves back onto the screen, to be misdrawn.

Whether a particular spot on the screen is specified with an absolute or a relative address is of no consequence. The sequence in which they are drawn is of no consequence. Each object is independent of all others, and exists at the place which was specified, in one way or other, by the command that created it. Relative addresses are provided for the sake of data compression. They are not an attempt to spare programs the need for the meagre intelligence required to convert between absolute and relative addresses; more intelligence than that will surely be required for other aspects of the graphics protocol. Nor are relative addresses intended to cause several objects to relocate together if one is "moved" or erased. Terminals are not expected to remember any relation between objects once they are drawn. Most will not be able to.

Although the cursor position on entry to graphics mode remains set from the last exit, it is wise to reinitialize it with a %GOMVA command before any long transfer, to limit the effects of lost output.

Commands:

Commands to draw an object always have counterparts which erase the same object. On a bit matrix terminal, erasure and drawing are almost identical operations. On a display list terminal, erasure involves searching the display list for an object with the specified characteristics and deleting it from the list. It is assumed that any terminal whose %TOERS bit is set can erase graphic objects.

The commands to draw objects run from 100 to 137, while those to erase run in a parallel sequence from 140 to 177. Other sorts of operations have command codes below 100. Meanwhile, the 20 bit in the command code says which type of addresses are used as arguments: if the 20 bit is set, absolute addresses are used. Graphics commands are given names starting with "%GO".

Graphics often uses characters. The %GODCH command is followed by a string of characters to be output, terminated by a zero. The characters must be single-position printing characters. On most terminals, this limits them to ASCII graphic characters. Terminals with %TOSAI set in the TTYOPT variable allow all characters 0-177. The characters are output at the current graphics cursor position (the lower left hand corner of the first character's rectangle being placed there), which is moved as the characters are drawn. The normal type-out cursor is not relevant and its position is not changed. The cursor position at which the characters are drawn may be in between the lines and columns used for normal type-out. The %GOECH command is similar to %GODCH but erases the characters specified in it. To clear out a row of character positions on a bit matrix terminal without having to respecify the text, a rectangle command may be used.

Example:

The way to send a simple line drawing is this:

```
%TDRST                  ;Reset all graphics modes.
%TDGRF                  ;Enter graphics.
%GOCLR                  ;Clear the screen.
%GOMVA xx yy            ;Set cursor.
%GODLA xx yy            ;Draw line from there.
<< repeat last two commands for each line >>
%TDNOP                  ;Exit graphics.
```

Graphics Input:

The %TRGIN bit in the right half of the SMARTS variable indicates that the terminal can supply a graphic input in the form of a cursor position on request. Sending a %GOGIN command to the terminal asks to read the cursor position. It should be followed by an argument character that will be included in the reply, and serve to associate the reply with the particular request for input that elicited it. The reply should have the form of a Top-Y character (code 4131), followed by the reply code character as just described, followed by an absolute cursor position. Since Top-Y is not normally meaningful as input, %GOGIN replies can be distinguished reliably from keyboard input. Unsolicited graphic input should be sent using a Top-X instead of a Top-Y, so that the program can distinguish them. Instead of a reply code, for which there is no need, the terminal should send an encoding of the buttons pressed by the user on his input device, if it has more than one.

Sets:

Terminals may define the concept of a "set" of objects. There are up to 200 different sets, each of which can contain arbitrarily many objects. At any time, one set is selected; objects drawn become part of that set, and objects erased are removed from it. Objects in a set other than the selected one cannot be erased without switching to the sets that contain them. A set can be made temporarily invisible, as a whole, without being erased or its contents forgotten; and it can then be made instantly visible again. Also, a whole set can be moved. A set has at all times a point identified as its "center", and all objects in it are actually remembered relative to that center, which can be moved arbitrarily, thus moving all the objects in the set at once. Before beginning to use a set, therefore, one should "move" its center to some absolute location. Set center motion can easily cause objects in the set to move off screen. When this happens, it does not matter what happens temporarily to those objects, but their "positions" must not be forgotten, so that undoing the set center motion will restore them to visibility in their previous positions. Sets are not easily implemented on bit matrix terminals, which should therefore ignore all set operations (except, for a degenerate interpretation in connection with blinking, if that is implemented). The %TQSET bit in the SMARTS variable of the terminal indicates that the terminal implements multiple sets of objects.

On a terminal which supports multiple sets, the %GOCLR command should empty all sets and mark all sets "visible" (perform a %GOVIS on each one). So should a %TDCLR SUPDUP command. Thus, any program which starts by clearing the screen will not have to worry about initializing the states of all sets.

Blinking:

Some terminals have the ability to blink objects on the screen. The command %GOBNK meaning make the current set blink. All objects in it already begin blinking, and any new objects also blink. %GOVIS or %TOINV cancels the effect of a %GOBNK, making the objects of the set permanently visible or invisible. %TQBNK indicates that the terminal supports blinking on the screen.

However, there is a problem: some intelligent bit matrix terminals may be able to implement blinking a few objects, if they are told in advance, before the objects are drawn. They will be unable to support arbitrary use of %GOBNK, however.

The solution to the problem is a convention for the use of %TOBNK which, together with degenerate definitions for set operations, makes it possible to give commands which reliably work on any terminal which supports blinking.

On a terminal which sets %TQBNK but not %TQSET, %GOBNK is defined to cause objects which are drawn after it to be drawn blinking. %GOSET cancels this, so following objects will be drawn unblinking. This is regardless of the argument to the %GOSET.

Thus, the way for a program to work on all terminals with %TQBNK, whether they know about sets or not, is: to write a bliniking picture, select some set other than your normal one (set 1 will do), do %GOBNK, output the picture, and reselect set 0. The picture will blink, while you draw things in set 0. To draw more blinking objects, you must reselect set 1 and do another %GOBNK. Simply reselecting set 1 will not work on terminals which don't really support sets, since they don't remember that the blinking objects are "in set 1" and not "in set 0".

Erasing a blinking object should make it disappear, on any terminal which implements blinking. On bit matrix terminals, blinking MUST always be done by XORing, so that the non-blinking background is not destroyed.

%GOCLS, on a terminal which supports blinking but not sets, should delete all blinking objects. Then, the convention for deleting all blinking objects is to select set 1, do a %GOCLS, and reselect set 0. This has the desired effect on all terminals. This definition of %GOCLS causes no trouble on non-set terminals, since %GOCLS would otherwise be meaningless to them.

To make blinking objects stop blinking but remain visible is possible with a %GOVIS on a terminal which supports sets. But in general the only way to do it is to delete them and redraw them as permanent.

Rectangles and XOR

Bit matrix terminals have their own operations that display list terminals cannot duplicate. First of all, they have XOR mode, in which objects drawn cancel existing objects when they overlap. In this mode, drawing an object and erasing it are identical operations. All %GOD.. commands act IDENTICALLY to the corresponding %GOE..'s. XOR mode is entered with a %GOXOR and left with a %GOIOR. Display list terminals will ignore both commands. For that reason, the program should continue to distinguish draw commands from erase commands even in XOR mode. %TQXOR indicates a terminal which implements XOR mode. XOR mode, when set, remains set even if graphics mode is left and re-entered. However, it is wise to re-specify it from time to time, in case output is lost.

Bit matrix terminals can also draw solid rectangles. They can thus implement the commands %GODRR, %GODRA, %GOERR, and %GOERA. A rectangle is specified by taking the current cursor position to be one corner, and providing the address of the opposite corner. That can be done with either a relative address or an absolute one. The %TQREC bit indicates that the terminal implements rectangle commands.

Of course, a sufficiently intelligent bit matrix terminal can provide all the features of a display list terminal by remembering display lists which are redundant with the bit matrix, and using them to update the matrix when a %GOMSR or %GOVIS is done. However, most bit matrix terminals are not expected to go to such lengths.

How Several Process Can Draw On One Terminal Without Interfering With Each Other:

If we define "input-stream state" information to be whatever information which can affect the action of any command, other than what is contained in the command, then each of the several processes must have its own set of input-stream state variables.

This is accomplished by providing the %GOPSH command. The %GOPSH command saves all such input-stream information, to be restored when graphics mode is exited. If the processes can arrange to output blocks of characters uninterruptibly, they can begin each block with a %GOPSH followed by commands to initialize the input-stream state information as they desire. Each block of graphics output should be ended by a %TDNOP, leaving the terminal in its "normal" state for all the other processes, and at the same time popping the what the %GOPSH pushed.

The input-stream state information consists of:

The cursor position
the state of XOR mode (default is OFF)
the selected set (default is 0)
the co-ordinate unit in use (physical dots, or virtual)
(default is physical)
whether output is going to the display screen or to a hardcopy device (default is to the screen)
what portion of the screen is in use
(see "Using Only Part of the Screen")
(default is all)

Each unit of input-stream status has a default value for the sake of programs that do not know that the information exists; the exception is the cursor position, since all programs must know that it exists. A %TDINI or %TDRST command should set all of the variables to their default values.

The state of the current set (whether it is visible, and where its center is) is not part of the input-stream state information, since it would be hard to say what it would mean if it were. Besides, the current set number is part of the input-stream state information, so different processes can use different sets. The allocation of sets to processes is the server host's own business.

Using Only Part of the Screen:

It is sometimes desirable to use part of the screen for picture and part for text. Then one may wish to clear the picture without clearing the text. On display list terminals, %GOCLR should do this. On bit matrix terminals, however, %GOCLR can't tell which bits were set by graphics and which by text display. For their sake, the %GOLMT command is provided. This command takes two cursor positions as arguments, specifying a rectangle. It declares that graphics will be limited to that rectangle, so %GOCLR should clear only that part of the screen. %GOLMT need not do anything on a terminal which can remember graphics output as distinct from text output and clear the former selectively, although it would be a desirable feature to process it even on those terminals.

%GOLMT can be used to enable one of several processes which divide up the screen among themselves to clear only the picture that it has drawn, on a bit matrix terminal. By using both %GOLMT and distinct sets, it is possible to deal successfully with almost any terminal, since bit matrix terminals will implement %GOLMT and display list terminals almost always implement sets.

The %TDCLR command should clear the whole screen, including graphics output, ignoring %GOLMT.

Errors:

In general, errors in graphics commands should be ignored.

Since the output and input streams are not synchronized unless trouble is taken, there is no simple way to report an error well enough for the program that caused it to identify just which command was invalid. So it is better not to try.

Errors which are not the fault of any individual command, such as running out of memory for display lists, should also be ignored as much as possible. This does NOT mean completely ignoring the commands that cannot be followed; it means following them as much as possible: moving the cursor, selecting sets, etc. as they specify, so that any subsequent commands which can be executed are executed as intended.

Extensions:

This protocol does not attempt to specify commands for dealing with every imaginable feature which a picture-drawing device can have. Additional features should be left until they are needed and well understood, so that they can be done right.

Storage of Graphics Commands in Files:

This can certainly be done. Since graphics commands are composed exclusively of the ASCII characters 0 - 177, any file that can hold ASCII text can hold the commands to draw a picture. This is less useful than you might think, however. Any program for editing, in whatever loose sense, a picture, will have its own internal data which determine the relationships between the objects depicted, and control the interpretation of the programs commands, and this data will all be lost in the SUPDUP graphics commands for displaying the picture. Thus, each such program will need to have its own format for storing pictures in files, suitable for that program's internal data structure. Inclusion of actual graphics commands in a file will be useful only when the sole purpose of the file is to be displayed.

Note: the values of these commands are represented as 8.-bit octal bytes. Arguments to the commands are in lower case inside angle brackets.

The Draw commands are:

Value	Name	Arguments
101	%GODLR	<p>
		Draw line relative, from the cursor to <p>.
102	%GODPR	<p>
		Draw point relative, at <p>.
103	%GODRR	<p>
		Draw rectangle relative, corners at <p> and at the current cursor position.
104	%GODCH	<string> <0>
		Display the chars of <string> starting at the current graphics cursor position.
121	%GODLA	<p>
		Draw line absolute, from the cursor to <p>. The same effect as %GODLR, but the arg is an absolute address.
122	%GODPA	<p>
		Draw point absolute, at <p>.
123	%GODRA	<p>
		Draw rectangle absolute, corners at <p> and at the current cursor position.

The Erase commands are:

Value	Name	Arguments
141	%GOELR	<p>
		Erase line relative, from the cursor to <p>.
142	%GOEPR	<p>
		Erase point relative, at <p>.
143	%GOERR	<p>
		Erase rectangle relative, corners at <p> and at the current cursor position.
144	%GOECH	<string> <0>
		Erase the chars of <string> starting at the current graphics cursor position.
161	%GOELA	<p>
		Erase line absolute, from the cursor to <p>.
162	%GOEPA	<p>
		Erase point absolute, at <p>.
163	%GOERA	<p>
		Erase rectangle absolute, corners at <p> and at the current cursor position.

The miscellaneous commands are:

Value	Name	Arguments
001	%GOMVR	<p> Move cursor to point <p>
021	%GOMVA	<p> Move cursor to point <p>, absolute address.
002	%GOXOR	 Turn on XOR mode. Bit matrix terminals only.
022	%GOIOR	 Turn off XOR mode.
003	%GOSET	<n> Select set. <n> is a 1-character set number, 0 - 177.
004	%GOMSR	<p> Move set origin to <p>. Display list terminals only.
024	%GOMSA	<p> Move set origin to <p>, absolute address.
006	%GOINV	 Make current set invisible.
026	%GOVIS	 Make current set visible.
007	%GOBNK	 Make current set blink. Canceled by %GOINV or %GOVIS.
010	%GOCLR	 Erase whole screen.
030	%GOCLS	 Erase entire current set (display list terminals).
011	%GOPSH	 Push all input-stream status information, to be restored when graphics mode is exited.
012	%GOVIR	 Start using virtual co-ordinates
032	%GOPHY	 Resume giving co-ordinates in units of dots.
013	%GOHRD	<n> Divert output to output subdevice <n>. <n>=0 reselects the main display screen.
014	%GOGIN	<n> Request graphics input (mouse, tablet, etc). <n> is the reply code to include in the answer.
015	%GOLMT	<p1> <p2> Limits graphics to a subrectangle of the screen. %GOCLR will clear only that area. This is for those who would use the rest for text.

Bits in the SMARTS Variable Related to Graphics:

Note: the values of these bits are represented as octal 36.-bit words, with the left and right 18.-bit halfword separated by two commas as in the normal PDP-10 convention.

Name	Value	Description
%TQGRF	000001,,0	terminal understands graphics protocol.
%TQSET	000002,,0	terminal supports multiple sets.
%TQREC	000004,,0	terminal implements rectangle commands.
%TQXOR	000010,,0	terminal implements XOR mode.
%TQBNK	000020,,0	terminal implements blinking.
%TQVIR	000040,,0	terminal implements virtual co-ordinates.
%TQWID	001700,,0	character width, in dots.
%TQHGT	076000,,0	character height, in dots.
%TRGIN	0,,400000	terminal can provide graphics input.
%TRGHC	0,,200000	terminal has a hard-copy device to which output can be diverted.